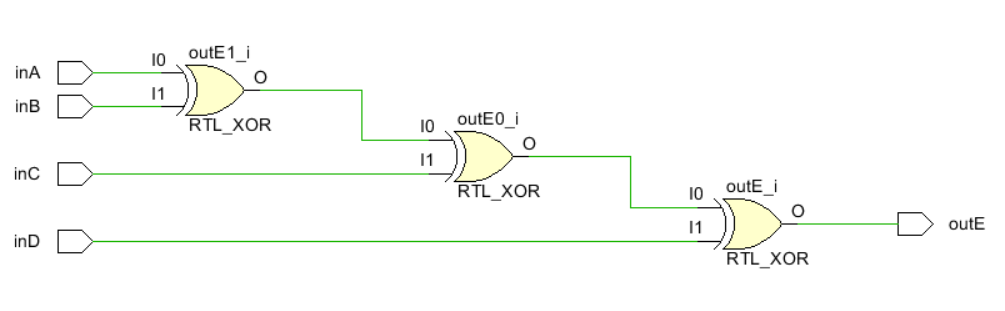
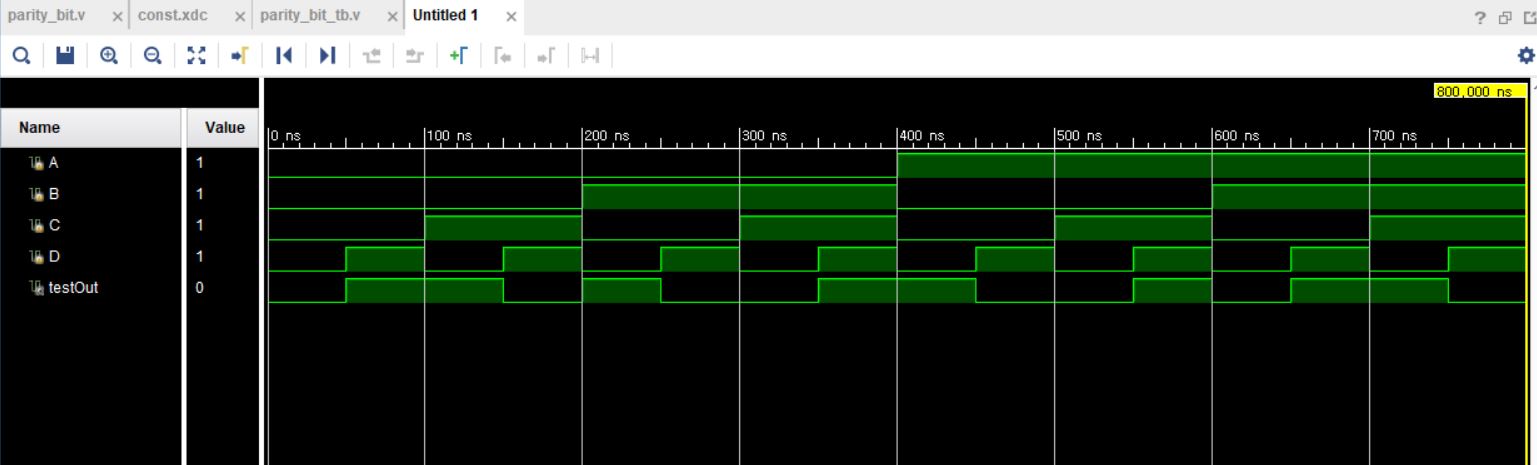
7주차 결과보고서

전공: 물리학과 학년: 3학년 학번: 20191286 이름: 김나현

1. Even Parity bit generator 및 checker의 simulation 결과 및 과정에 대해서 설명하시오(진리표 작성 및 k-map 포함).



<Figure 1> Even parity bit generator schematic diagram



<Figure 2> Even parity bit generator simulation 결과

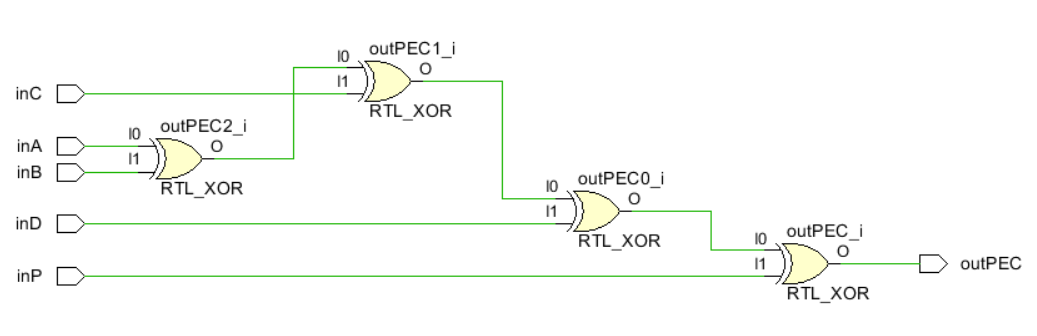
<Table 1> Even parity bit generator 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | Parity bit |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

<Table 1-1> Even parity bit generator 카르노맵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CD\AB | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 0 | 1 |
| 01 | 1 | 0 | 1 | 0 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |

4 bit 데이터를 전송하려고 할 때, even parity bit generator를 이용하면 전송하려는 1 bit, 4 개의 데이터, A, B, C, D 중 1인 입력의 개수가 짝수일 때는 출력, 즉 even parity bit가 0이 되고 1인 입력의 개수가 홀수일 때는 출력이 1이 됩니다. 따라서, even parity bit generator는 n개의 입력 데이터에 대해 n-1개의 XOR 게이트를 이용하여 출력인 even parity bit를 결정합니다. Verilog로 even parity bit generator를 코딩한 후, test bench를 이용해 simulation 결과를 확인해보면 위의 <Figure 2>와 같고, 이를 진리표로 작성하면 <Table 1>과 같이 됩니다. 이는 입력 중 1인 입력의 개수가 홀수일 때만 even parity bit가 1이 된다는 even parity bit generator의 이론적인 특성과 일치하므로 올바르게 Verilog 코딩을 한 것이라는 것을 알 수 있습니다. Even parity bit generator가 XOR 게이트를 이용하여 구현된다는 것을 이미 알고 있지만 그렇지 않은 경우, <Table 1>을 통해 <Table 1-1>과 같은 카르노맵을 작성하여 P=A’BC’D’+AB’C’D’+A’B’C’D+ABC’D+ABC’D+A’BCD+ AB’CD+A’B’CD’+ABCD’, 즉 P=A⊕B⊕C⊕D라는 논리 함수로 even parity bit generator의 출력을 나타낼 수 있습니다.



<Figure 3> Even parity bit checker schematic diagram



<Figure 4> Even parity bit checker simulation 결과

<Table 2> Even parity bit checker 진리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | D | P | PEC |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

<Table 2-1> Even parity bit checker 카르노맵

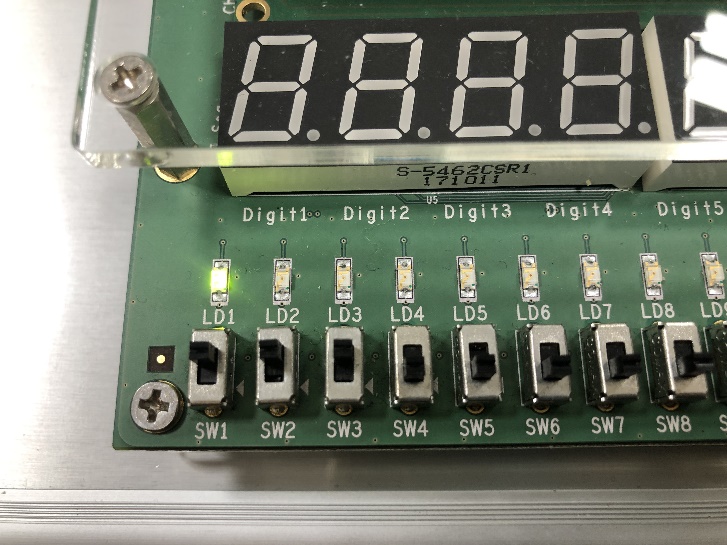
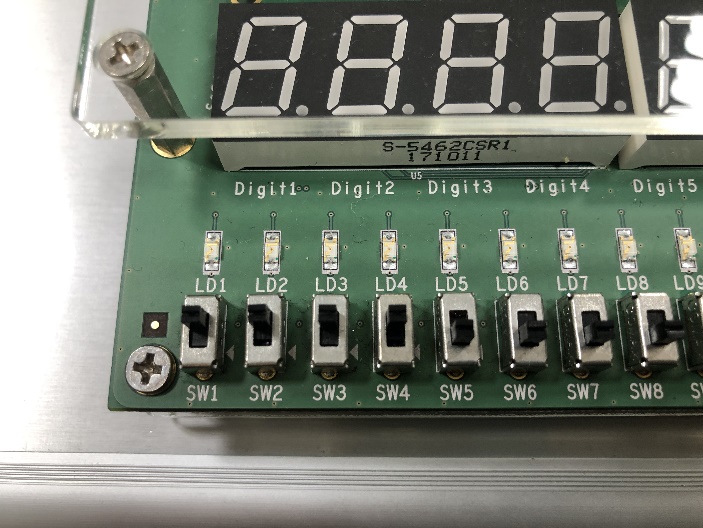
A=0일 때,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DP\BC | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 0 | 1 |
| 01 | 1 | 0 | 1 | 0 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |

A=1일 때,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DP\BC | 00 | 01 | 11 | 10 |
| 00 | 1 | 0 | 1 | 0 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 0 |
| 10 | 0 | 1 | 0 | 1 |

4 bit 데이터를 전송하려고 할 때, even parity bit를 추가하여 이용하면 전송하면 even parity bit checker에서는 parity bit를 포함하여, A, B, C, D, P, 5 개의 데이터 bit 중 1인 입력의 개수가 홀수일 때는 에러를 의미하는 1을, 1인 입력의 개수가 짝수일 때는 에러가 발생하지 않음을 의미하는 0을 출력하게 됩니다. 따라서, even parity bit checker도 even parity bit generator와 동일하게 parity bit를 포함하여 n개의 입력 데이터에 대해 n-1개의 XOR 게이트를 이용하여 출력인 PEC(parity error check)를 결정합니다. Verilog로 even parity bit checker를 코딩한 후, test bench를 이용해 simulation 결과를 확인해보면 위의 <Figure 4>와 같고, 이를 진리표로 작성하면 <Table 2>과 같이 됩니다. 이는 입력 중 1인 입력의 개수가 홀수일 때만 PEC가 1이 된다는 even parity bit checker의 이론적인 특성과 일치하므로 올바르게 Verilog 코딩을 한 것이라는 것을 알 수 있습니다. 우리는 Even parity bit checker가 XOR 게이트를 이용하여 구현된다는 것을 이미 알고 있지만 even parity bit checker가 어떻게 구현되는지 알려주지 않고 진리표만을 준 경우, <Table 2>를 통해 <Table 2-1>과 같이 카르노맵을 작성하여 PEC=A’(B’CD’P’+BC’D’P’+B’C’D’P+BCD’P+BCD’P+ B’CDP+BC’DP+B’C’DP’+BCDP’)+A(B’C’D’P’+BCD’P’+B’CD’P+BC’D’P+B’C’DP+BCDP+B’CDP’+BC’DP’), 즉 PEC=A⊕B⊕C⊕D⊕P라는 논리 함수로 even parity bit checker의 출력을 나타낼 수 있습니다.

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

<Figure 2-1> FPGA을 통해 확인한 Even parity bit generator 동작

위의 <Figure 2-1>은 위, 왼쪽에서부터 A, B, C, D의 입력 중 1인 입력이 0개, 1개, 2개, 3개, 4개일 때의 모습입니다. 4개의 입력 중 1인 입력의 개수가 홀수일 때만 LD1에 불이 들어오는 모습을 확인할 수 있는데, LD1에 불이 들어오는 것은 출력이 1임을 의미하므로 even parity bit generator는 n개의 입력 중 1인 입력의 개수가 홀수일 때 1을 출력한다는 것을 실험적으로 확인할 수 있었습니다.

<Figure 4-1> FPGA을 통해 확인한 Even parity bit checker 동작텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

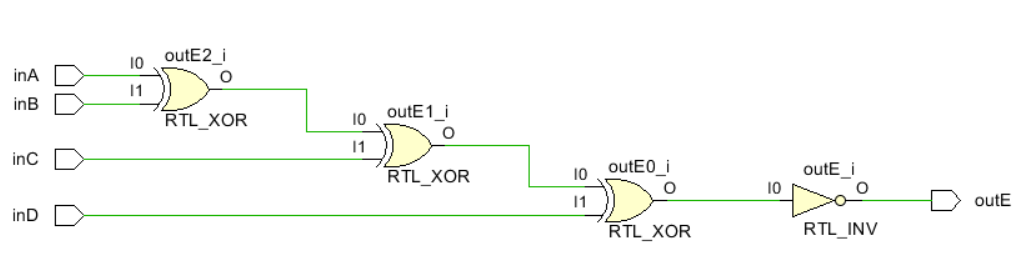
자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

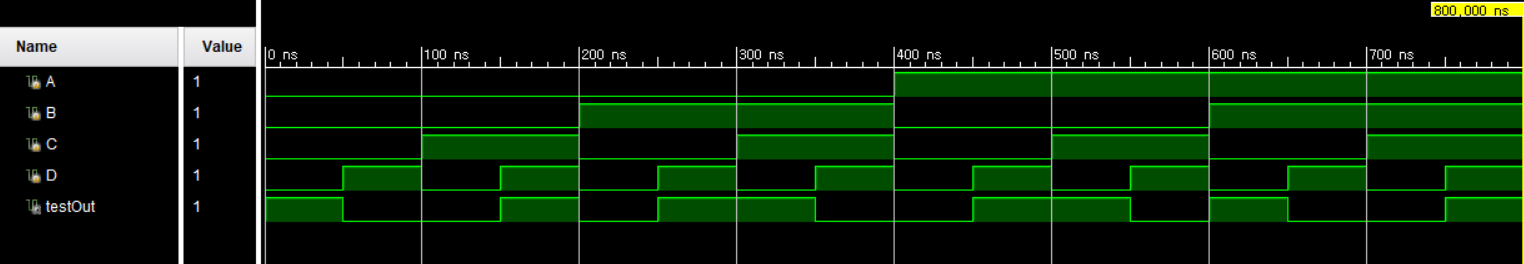
자동 생성된 설명

위의 <Figure 4-1>은 위, 왼쪽에서부터 A, B, C, D, P의 입력 중 1인 입력이 0개, 1개, 2개, 3개, 4개, 5개일 때의 모습입니다. 5개의 입력 중 1인 입력의 개수가 홀수일 때만 LD1에 불이 들어오는 모습을 확인할 수 있는데, LD1에 불이 들어오는 것은 출력이 1임을 의미하므로 even parity bit checker는 n개의 입력 중 1인 입력의 개수가 홀수일 때 1을 출력하여 에러 발생을 표시한다는 것을 실험적으로 확인할 수 있었습니다.

1. Odd Parity bit generator 및 checker의 simulation 결과 및 과정에 대해서 설명하시오(진리표 작성 및 k-map 포함).



<Figure 5> Odd parity bit generator schematic diagram



<Figure 6> Odd parity bit generator simulation 결과

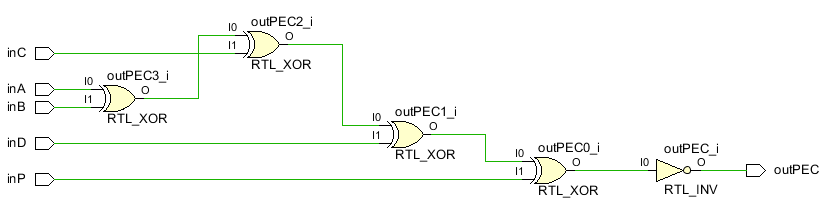
<Table 3> Odd parity bit generator 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | Parity bit |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

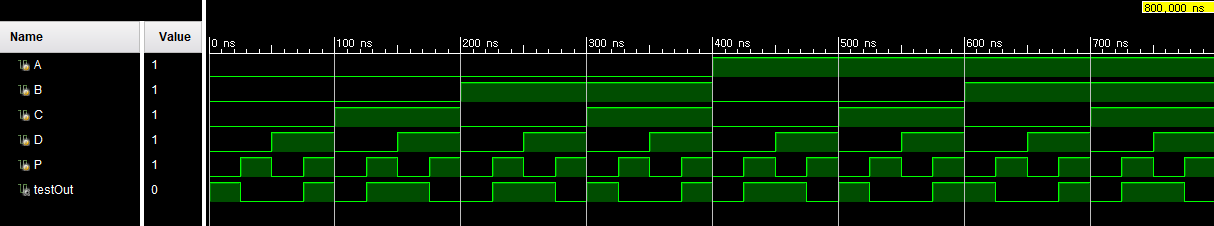
<Table 3-1> Odd parity bit generator 카르노맵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CD\AB | 00 | 01 | 11 | 10 |
| 00 | 1 | 0 | 1 | 0 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 0 |
| 10 | 0 | 1 | 0 | 1 |

4 bit 데이터를 전송하려고 할 때, odd parity bit generator를 이용하면 전송하려는 1 bit, 4 개의 데이터, A, B, C, D 중 1인 입력의 개수가 홀수일 때는 출력, 즉 even parity bit가 0이 되고 1인 입력의 개수가 짝수일 때는 출력이 1이 됩니다. 따라서, odd parity bit generator는 n개의 입력 데이터에 대해 n-1개의 XOR 게이트와 한 개의 NOT 게이트를 이용하여 출력인 odd parity bit를 결정합니다. Verilog로 odd parity bit generator를 코딩한 후, test bench를 이용해 simulation 결과를 확인해보면 위의 <Figure 6>과 같고, 이를 진리표로 작성하면 <Table 3>과 같이 됩니다. 이는 입력 중 1인 입력의 개수가 짝수일 때만 odd parity bit가 1이 된다는 odd parity bit generator의 이론적인 특성과 일치하므로 올바르게 Verilog 코딩을 한 것이라는 것을 알 수 있습니다. 우리는 Odd parity bit generator가 XOR 게이트와 NOT 게이트를 이용하여 구현된다는 것을 이미 알고 있지만 그렇지 않은 경우, Odd parity bit generator 진리표인 <Table 3>을 통해 <Table 3-1>과 같은 카르노맵을 작성하여 P=A’B’C’D’+ ABC’D’+A’BC’D+AB’C’D+A’B’CD+ABCD+A’BCD’+AB’CD’, 즉 P=(A⊕B⊕C⊕D)’라는 논리 함수로 odd parity bit generator의 출력을 나타낼 수 있습니다.



<Figure 7> Odd parity bit checker schematic diagram



<Figure 8> Odd parity bit checker simulation 결과

<Table 4> Odd parity bit checker 진리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | D | P | PEC |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

<Table 4-1> Odd parity bit checker 카르노맵

A=0일 때,

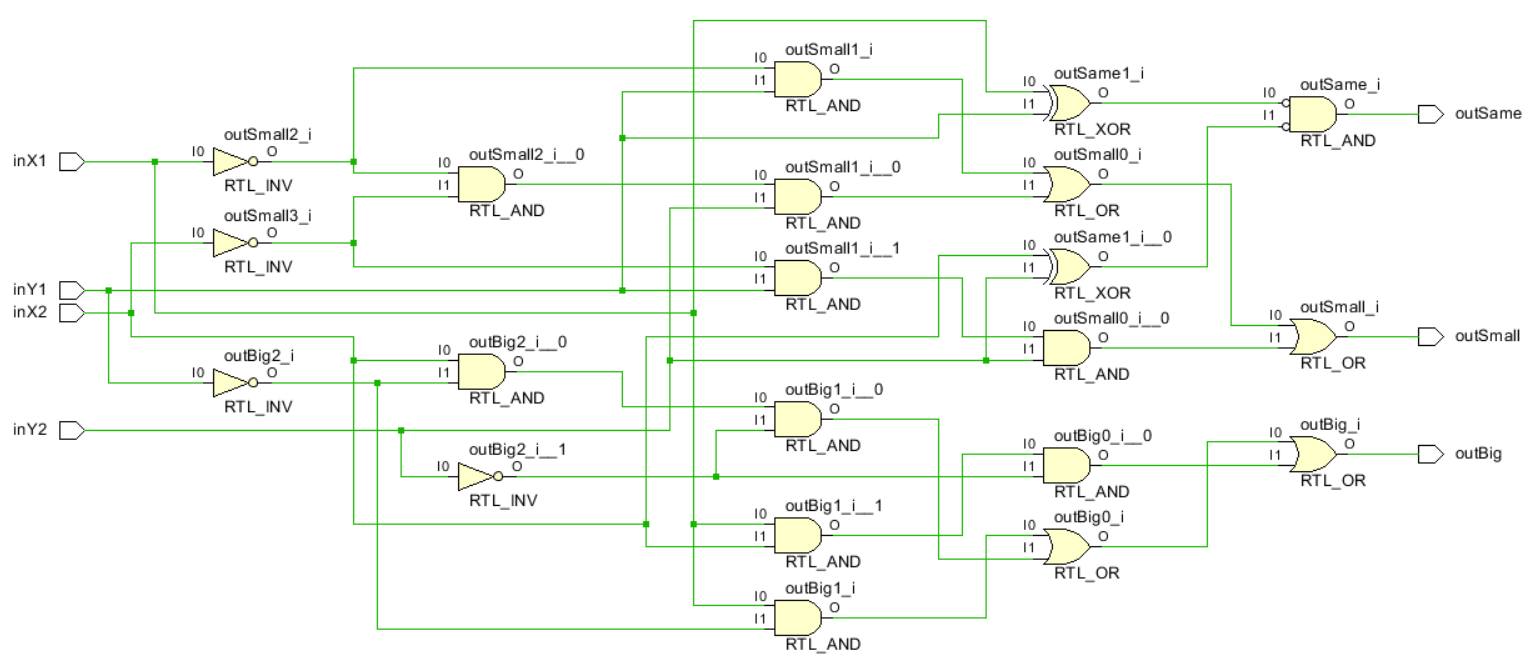
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DP\BC | 00 | 01 | 11 | 10 |
| 00 | 1 | 0 | 1 | 0 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 0 |
| 10 | 0 | 1 | 0 | 1 |

A=1일 때,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DP\BC | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 0 | 1 |
| 01 | 1 | 0 | 1 | 0 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |

4 bit 데이터를 전송하려고 할 때, odd parity bit를 추가하여 이용하면 전송하면 odd parity bit checker에서는 parity bit를 포함하여, A, B, C, D, P, 5 개의 데이터 bit 중 1인 입력의 개수가 짝수일 때는 에러를 의미하는 1을, 1인 입력의 개수가 홀수일 때는 에러가 발생하지 않음을 의미하는 0을 출력하게 됩니다. 따라서, odd parity bit checker도 odd parity bit generator와 동일하게 parity bit를 포함하여 n개의 입력 데이터에 대해 n-1개의 XOR 게이트와 한 개의 NOT 게이트를 이용하여 출력인 PEC(parity error check)를 결정합니다. Verilog로 odd parity bit checker를 코딩한 후, test bench를 이용해 simulation 결과를 확인해보면 위의 <Figure 8>과 같고, 이를 진리표로 작성하면 <Table 4>와 같이 됩니다. 이는 입력 중 1인 입력의 개수가 짝수일 때만 PEC가 1이 된다는 odd parity bit checker의 이론적인 특성과 일치하므로 올바르게 Verilog 코딩을 한 것이라는 것을 알 수 있습니다. 우리는 odd parity bit checker가 XOR 게이트와 NOT 게이트를 이용하여 구현된다는 것을 이미 알고 있지만 odd parity bit checker가 어떻게 구현되는지 알려주지 않고 진리표만을 준 경우, <Table 4>를 통해 <Table 4-1>과 같이 카르노맵을 작성하여 PEC=A(B’CD’P’+BC’D’P’+ B’C’D’P+BCD’P+BCD’P+B’CDP+BC’DP+B’C’DP’+BCDP’)+A’(B’C’D’P’+BCD’P’+B’CD’P+BC’D’P+B’C’DP+BCDP+B’CDP’+BC’DP’), 즉 PEC=(A⊕B⊕C⊕D⊕P)’라는 논리 함수로 odd parity bit checker의 출력을 나타낼 수 있습니다.

1. 2-bit binary comparator 결과 및 과정에 대해서 설명하시오(진리표 작성 및 k-map 포함).



<Figure 9> 2-bit comparator schematic diagram



<Figure 10> 2-bit comparator simulation 결과

<Table 5> 2-bit comparator 진리표

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X1 | X2 | Y1 | Y2 | outBig | outSame | outSmall |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

<Table 5-1> 2-bit comparator outBig 진리표

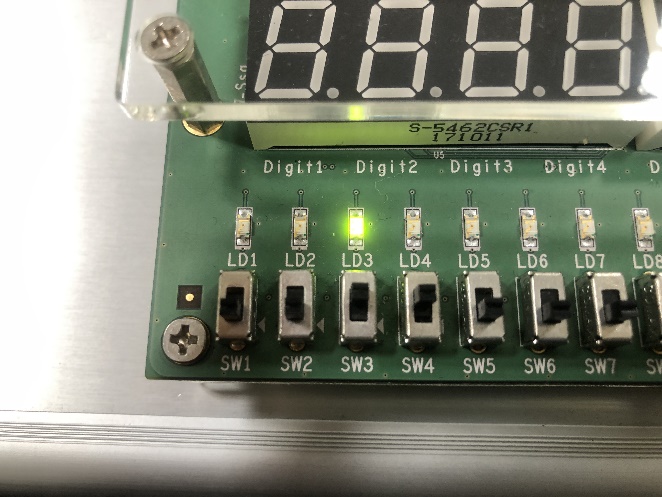
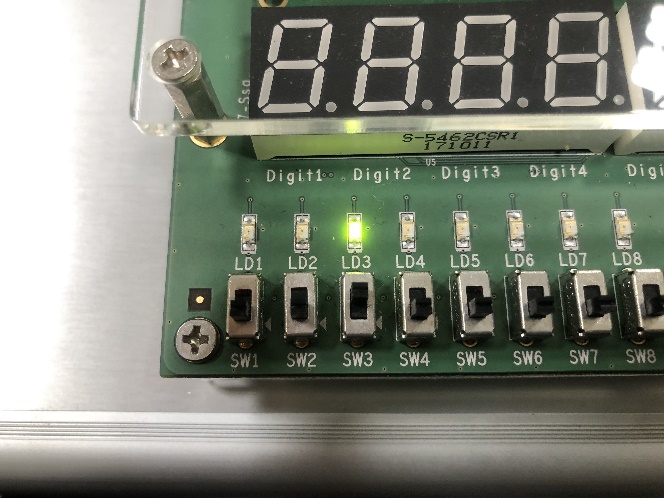
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Y1Y2\X1X2 | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 1 | 1 |
| 01 | 0 | 0 | 1 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 1 | 0 |

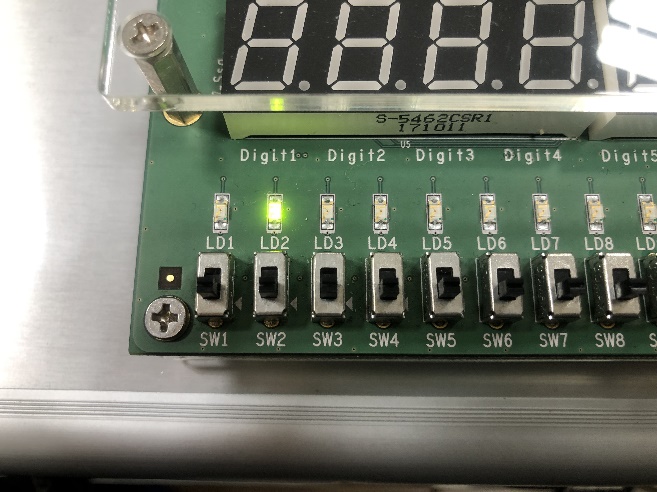
<Table 5-2> 2-bit comparator outSame 진리표

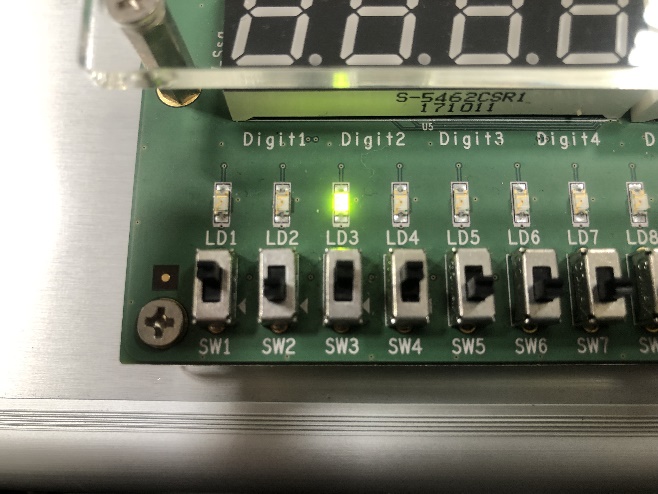
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Y1Y2\X1X2 | 00 | 01 | 11 | 10 |
| 00 | 1 | 0 | 0 | 0 |
| 01 | 0 | 1 | 0 | 0 |
| 11 | 0 | 0 | 1 | 0 |
| 10 | 0 | 0 | 0 | 1 |

<Table 5-3> 2-bit comparator outSmall 진리표

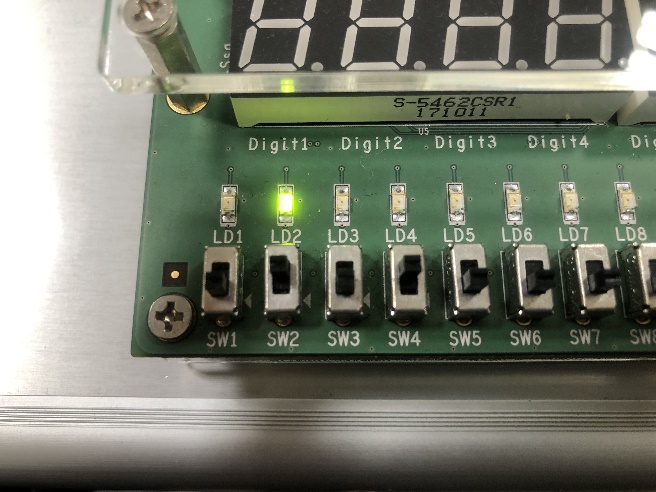
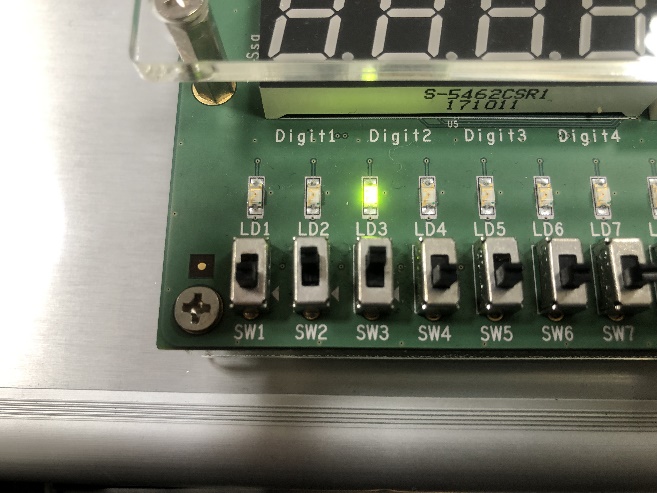
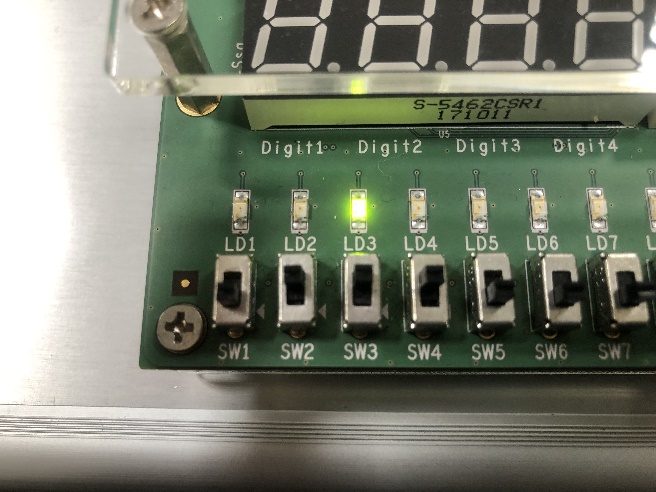
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Y1Y2\X1X2 | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 0 | 0 | 0 |
| 11 | 1 | 1 | 0 | 1 |
| 10 | 1 | 1 | 0 | 0 |

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명2비트 비교기는 2비트로 이루어진 두 개의 이진수를 비교하여 outBig, outSame, outSmall, 즉 3개의 1비트를 출력하는 논리 회로입니다. Verilog을 통해 2-bit comparator을 코딩한 후 schematic을 확인하면 위의 <Figure 9>와 같고 test bench를 만들어 simulation 결과를 확인하면 <Figure 10>과 같습니다. Simulation 결과를 통해 2-bit comparator의 진리표를 작성해보면 위의 <Table 5>와 같습니다. 이때, X와 Y를 비교하는 것으로 outBig이 1인 경우, X가 Y보다 크다는 뜻이고 outSmall이 1이면 X가 Y보다 작다는 것입니다. X는 X1, X2 두 개의 비트로 이루어져 있고, Y는 Y1, Y2 두 개의 비트로 이루어져 있습니다. 진리표를 보고 각 출력에 대한 논리 함수를 알아보기 위해 카르노맵을 그리면 위의 <Table 5-1>, <Table 5-2>, <Table 5-3>과 같고 논리 함수를 나타내보면 outBig은 outBig=(X1)(Y1)’+(X2)(Y1)’(Y2)’+(X1)(X2)(Y2)’, outSame은 outSame=((X1^Y1)(X2^Y2))’, outSmall은 outSmall=(X1)’(Y1)+(X1)’(X2)’(Y2)+(X2)’(Y1)(Y2)로 나타낼 수 있습니다.

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명<Figure 10-1> FPGA를 통해 확인한 2-bit comparator의 동작

위의 <Figure 10-1>은 위, 왼쪽에서부터 X1, X2, Y1, Y2의 입력이 0000부터 1111까지 하나씩 순서대로 증가할 때, 2-bit comparator의 출력을 확인한 모습입니다. 불이 들어오는 곳은 LD1, LD2, LD3로 총 세 곳인데, LD1은 outBig을, LD2는 outSame을, LD3는 outSmall을 의미합니다. LD2는 X1X2와 Y1Y2가 00, 01, 10, 11로 서로 같을 때 불이 들어왔고, LD1은 X1X2가 Y1Y2보다 클 때, LD3는 X1X2가 Y1Y2보다 작을 때 불이 들어온 것을 실험적으로 확인할 수 있었습니다.

1. 결과 검토 및 논의 사항

Even parity bit generator와 checker는 XOR 게이트를 이용하여 구현할 수 있었고, 4개 이상의 입력에 대해서도 XOR 게이트를 추가하는 방식으로 어렵지 않게 구현할 수 있겠다는 생각이 들었습니다.

Odd parity bit generator와 odd parity bit checker는 even parity bit generator와 even parity bit checker에 NOT 게이트를 붙여서 구현할 수 있었는데 굳이 특별한 이유가 없다면 odd parity bit를 이용하는 것보다 even parity bit를 이용하는 것이 verilog 코딩도 편하고 실제로 회로를 구현할 때 사용하는 게이트도 적어 경제적이라는 생각이 들었습니다.

2-bit comparator는 지난 실험에서 이미 다룬 적 있는 1-bit comparator보다 조금은 복잡했지만 2-bit comparator가 어떻게 동작하는지를 미리 알고 있기 때문에 카르노맵을 사용하여 어렵지 않게 Verilog 코딩할 수 있었습니다.

1. 추가 이론 조사 및 작성

Odd parity bit checker을 만들 때, 5개의 1비트 입력 중 1인 입력의 개수가 홀수일 때 0을, 짝수일 때 1을 출력해야 한다는 사실로부터 입력 중 1인 입력의 개수가 짝수일 때 1을 출력하는 XNOR 게이트로 회로를 구성하면 될 것이라고 생각하였습니다. 하지만 XNOR 게이트로 회로를 구성하고 simulation 결과를 확인해보았더니 원하는 결과가 나오지 않았습니다. Odd parity bit generator를 구현할 때는 XNOR 게이트로 회로를 구현하였을 때, 원하는 결과가 나왔는데 똑같은 방법으로 구현하였음에도 Odd parity bit checker의 경우에는 원하는 결과가 나오지 않았습니다.

Odd parity bit checker는 원래 5개의 입력 A, B, C, D, P가 모두 0일 때, odd parity bit checker는 에러를 나타내는 1을 출력해야 하지만 XNOR 게이트로 odd parity bit checker회로를 구현하였을 때는 0을 출력하였습니다. 이는 A와 B가 XNOR 연산을 통해 1을 출력하고, 1과 C가 XNOR 연산을 통해 0을 출력하고 0과 D는 XNOR 연산을 통해 1을 출력하고 마지막으로 1과 P는 XNOR 연산을 통해 0을 출력하기 때문이었습니다. 따라서, XNOR 게이트를 이용해 odd parity bit checker을 구현하면 안 되고 이 경우에는 odd parity bit generator를 XNOR 게이트로 구현한 것이 맞았지만 입력의 개수가 짝수가 아니라 홀수이면 원하는 결과를 얻을 수 없기 때문에 입력의 개수와 상관없이 사용하기 위해 even parity bit generator와 even parity bit checker에 NOT 게이트를 추가하여 odd parity bit generator와 odd parity bit checker을 구현해야 한다는 결론을 내릴 수 있었습니다.